# Une étude empirique des heuristiques de branchement pour le problème MAX-SAT

Frédéric Lardeux, Frédéric Saubion, Jin-Kao Hao

# Une étude empirique des heuristiques de branchement pour le problème MAX-SAT

**Frédéric Lardeux  Frédéric Saubion  Jin-Kao Hao**

LERIA, Université d'Angers,
2 Bd Lavoisier,
49045 Angers Cedex 01

{lardeux,saubion,hao}@info.univ-angers.fr

**Résumé**

MOMS et Jeroslow-Wang sont deux des heuristiques les plus utilisées pour ordonner les variables dans les algorithmes traitant les problèmes SAT et MAX-SAT. Toutefois, leurs comportements ne sont pas très bien compris dans le contexte du problème MAX-SAT. Dans ce papier, nous nous sommes intéressés à ces deux heuristiques ainsi qu'à l'heuristique Dynamic- weighting pour la résolution du problème MAX-SAT. Nous montrons empiriquement que leurs comportements dans un algorithme de type Branch and Bound dépend de la borne supérieure initiale ainsi que de mécanismes spécifiques comme la propagation unitaire ou encore l'estimation de la borne inférieure. À partir de ces résultats, nous avons proposé une combinaison intuitive des heuristiques MOMS and Jeroslow-Wang qui fournit des performances intéressantes.

**Abstract**

MOMS and Jeroslow-Wang are two mostly used heuristics for variable ordering in SAT and MAX-SAT solving. However, their behaviors are not well understood in the context of the MAX-SAT problem. In this paper, we are interested in these two heuristics as well the Dynamic-weighting heuristics for solving MAX-SAT. We show empirically that their behaviors within a basic Branch&Bound algorithm depend on the initial upper bound and on specfic mechanisms as the unit propagation or the lower bound estimation. Based on this result, we propose an intuitive combination of MOMS and Jeroslow-Wang heuristics that provides promising performance.

## 1 Introduction

Le problème de satisfiabilité (SAT) [6] consiste à trouver une affectation booléenne validant une formule en logique propositionnelle. Une instance de ce problème est définie par un ensemble de variables booléennes $\mathcal{X} = \{x_1, ..., x_n\}$ et une formule booléenne $\phi$: $\{0,1\}^n \rightarrow \{0,1\}$. Un littéral est une variable ou sa négation. Une affectation est une fonction $v$: $\mathcal{X} \rightarrow \{0,1\}$. La formule $\phi$ est en forme normale conjonctive (CNF) si c'est une conjonction de clauses où une clause est une disjonction de littéraux. Dans ce papier, $\phi$ est supposée être en CNF donc une formule peut être vue comme un ensemble de variables $\mathcal{X}$ et un ensemble de clauses $\mathcal{C}$. Le problème de satisfiabilité maximum (MAX-SAT) est la version optimisation de SAT qui consiste à trouver une affectation qui satisfait le nombre maximum de clauses dans $\phi$.

Beaucoup d'algorithmes complets ont été développés pour résoudre le problème SAT [12, 10, 14, 5, 15] et sont presque tous basés sur la procédure Davis-Putnam-Logemann-Loveland (DPLL) [3]. Ils diffèrent essentiellement par leurs heuristiques de branchement. Ils fournissent de bons résultats pour SAT mais sont impuissants face au problème MAX-SAT.

Contrairement au problème SAT, peu d'algorithmes ont été proposés pour MAX-SAT. L'un des premiers fut celui présenté dans [9, 2]. Cet algorithme est un algorithme de type Branch and Bound (B&B) classique combinant une extension de DPLL avec l'algorithme de recherche locale GSAT [17]. Ici, GSAT est utilisé pour obtenir une borne supérieure initiale (que l'on espère bonne). Inspirés par cette idée, la plupart des algorithmes complets traitant le problème MAX-SAT [1, 22, 20] sont basés sur cette com-

binaison améliorée par des mécanismes spécifiques pour B&B : la propagation unitaire (UP) et l'estimation de la borne inférieure (LB2). Les heuristiques de branchements utilisées pour le problème SAT sont reprises par les algorithmes pour le problème MAX-SAT. Malheureusement, aucune étude approfondie n'a été effectuée pour caractériser leur comportement dans le contexte de la résolution de MAX-SAT.

Ce manque d'études limite certainement les performances des algorithmes complets pour MAX-SAT. Dans ce papier, nous proposons une étude empirique de deux des heuristiques les plus connues pour SAT (MOMS [4] et Jeroslow-Wang (JW) [8]) et d'une heuristique spécifique pour MAX-SAT (l'heuristique Dynamic-weighting [20]). Nous caractérisons leurs comportements par rapport à différents paramètres et proposons une heuristique intuitive combinant MOMS et JW. Nous montrons que l'utilisation du mécanisme UP améliore les résultats de toutes les heuristiques alors que le mécanisme LB2 n'améliore que ceux de l'heuristique JW.

Ce papier suit le plan suivant. La prochaine section présente l'algorithme B&B pour MAX-SAT. Ensuite, les définitions des heuristiques de branchement sont fournies. Enfin, une analyse précise du comportement des heuristiques de branchement ainsi qu'une hybridation intuitive de ces dernières sont proposées.

## 2 Algorithme de type Branch and Bound appliqué au problème MAX-SAT

Le B&B pour MAX-SAT peut être vu comme une extension de l'algorithme bien connu DPLL. Le principe général se fonde toujours sur l'exploration d'un arbre de recherche mais plusieurs techniques spécifiques sont introduites pour MAX-SAT.

### 2.1 Branch and Bound de base

L'algorithme B&B explore un arbre de recherche afin de trouver l'affectation fournissant le plus petit nombre de clauses fausses pour une formule CNF donnée. B&B (Algorithme 1) commence par sélectionner une variable $v$ non valuée grâce à une heuristique de branchement donnée. Cette variable est ensuite instanciée et la formule est simplifiée en fonction de cette valuation : quand $v$ est mise à $vrai$ (resp. $faux$), les clauses contenant le littéral $v$ (resp. $\neg v$) sont supprimées et toutes les occurrences du littéral $\neg v$ (resp. $v$) sont effacées des clauses les contenant. Ce processus est connu sous le nom de règle du littéral unique [11]. Une clause dont tous les littéraux ont été effacés (clause vide) correspond à une clause fausse. Si le nombre de clauses fausses obtenues avec cette valuation est supérieur à $ub$ (la borne supérieur représentant le nombre de clauses fausses obtenues par la meilleure affectation trouvée depuis le début de la recherche) alors aucune affectation construite à partir de ce nœud ne peut fournir un nombre de clauses fausses plus petit que $ub$. La branche correspondante est donc coupée. À ce moment, soit la variable courante est flippée ($true$ à $false$ ou $false$ à $true$), soit, si les deux valeurs de vérité possibles ont été testées, l'algorithme effectue un backtrack (la valeur de vérité de la variable courante est effacée et la variable précédente est inspectée).

Quand une affectation complète dont le nombre de clauses vides correspondant est inférieur à $ub$ est trouvée, alors $ub$ est mise à jour avec cette nouvelle valeur. Le processus ne s'arrête que quand tous les backtracks possibles ont été effectués. Le nombre minimum de clauses fausses pour la formule initialement donnée est la valeur courante de $ub$.

Au début de la recherche, $ub$ doit être initialisée. Au moins deux possibilités existent. La première consiste à démarrer avec une valeur égale au nombre de clauses de la formule mais une telle initialisation mène à une large exploration de l'arbre de recherche. Pour réduire cette exploration, la seconde technique consiste à utiliser un algorithme approché [17, 16] qui retourne une affectation engendrant peu de clauses fausses. Il est à noter que, étant donné la bonne efficacité de ces algorithmes, la valeur initiale $ub$ obtenue de cette manière est souvent de très bonne qualité. En fait, dans beaucoup de cas, cette valeur $ub$ correspond à la meilleure valeur possible.

**Algorithme 1** : Branch and Bound

**Données** : une formule $\phi$ en CNF, une borne supérieure $ub$
**Résultat** : le nombre minimum de clauses fausses

```
1  début
2      si φ = ∅ ou φ ne contient que des clauses vides alors
3          Retourner le nombre de clauses vides de φ;
4      fin
5      si le nombre de clauses vides de φ ≥ ub alors
6          Retourner ∞;
7      fin
8      x = variable sélectionnée par une heuristique;
9      ub = min(ub, Branch_and_Bound(φ_¬v, ub));
10     Retourner
       min(ub, Branch_and_Bound(φ_v, ub));
11 fin
```

Plusieurs techniques ont été développées pour le problème MAX-SAT afin d'améliorer l'algorithme général.

### 2.2 Propagation Unitaire

Le mécanisme de propagation unitaire (UP) agit sur les clauses unitaires (clauses ne contenant qu'un seul littéral).

Pour le problème SAT, ce littéral doit être mis à vrai et ensuite la simplification de la formule devient facile en utilisant la règle du littéral unique. Si de nouvelles clauses unitaires apparaissent, cette opération est répétée jusqu'à ce que l'état de saturation (point fixe) soit atteint ou que deux clauses unitaires $(v)$ et $(\neg v)$ apparaissent assurant ainsi la génération d'une clause vide par la règle de résolution classique. Quand un backtrack est réalisé, les variables valuées durant le mécanisme de propagation unitaire ne doivent pas être flippée car sinon une clause vide apparaîtrait.

Pour le problème MAX-SAT, la propagation unitaire n'est pas systématique [18]. Une clause unitaire $(v)$ (resp. $(\neg v)$) n'est mise à vrai (resp. faux) que si la différence entre la borne supérieure et le nombre de clauses vides (noté $cv_\phi$) est inférieure au nombre de clauses unitaires $(v)$ (resp. $(\neg v)$) et que le nombre de clauses unitaires $(\neg v)$ (resp. $(v)$) est supérieur à la différence entre la borne supérieure et le nombre de clauses vides. Le nombre de clauses unitaires contenant un littéral donné est obtenu par la fonction $cu$.

Dans l'algorithme 1, la propagation unitaire est insérée ligne 7 :

**Algorithme 2** : Propagation Unitaire

```
1 si ∃ une variable v telle que (((cv_φ + cu(v) ≥ ub) ∧
  (cv_φ + cu(¬v) < ub)) ∨ ((cv_φ + cu(¬v) ≥ ub) ∧
  (cv_φ + cu(v) < ub)) alors
2   si cv_φ + cu(v) ≥ ub alors
3     Retourner
      min(ub, Branch_and_Bound(φ_v, ub));
4   fin
5   si cv_φ + cu(¬v) ≥ ub alors
6     Retourner
      min(ub, Branch_and_Bound(φ_¬v, ub));
7   fin
8 fin
```

D'autres mécanismes de propagations unitaires ont été récemment proposés dans [20]. Ils utilisent une formulation en programmation non linéaire du problème MAX-SAT.

Nous allons maintenant introduire d'autres techniques qui ont été spécifiquement développées pour les algorithmes de type B&B.

### 2.3 Estimation de la Borne Inférieure

La borne inférieure représente le nombre minimum de clauses vides engendrées par l'affectation courante. À chaque étape du Branch and Bound, ce nombre de clauses vides peut être borné. Une borne inférieure triviale, appelée $LB1$, est $LB1(\phi) =$ *le nombre de clauses vides de la formule* $\phi$ et une estimation du nombre minimum de clauses vides est :

$$LB1(\phi) \leq nombre\ minimum\ de\ clauses\ vides\ de\ \phi \leq ub$$

Pour couper des branches de l'arbre de recherche plus rapidement, une borne inférieure élevée est nécessaire [18, 1, 22, 19]. L'idée consiste donc à examiner les clauses dans le but de déterminer combien d'entre elles seront vides dans les prochaines valuations. Par exemple, si trois clauses unitaires contiennent la même variable mais deux sous la forme d'un littéral positif et l'autre sous celle d'un littéral négatif, au moins une des trois clauses sera vide au meilleur des cas (deux au pire) quand toutes les variables seront valuées. Cette estimation peut être généralisée avec la fonction suivante [1] :

$$LB2(\phi) = \text{nombre actuel de clauses vides de } \phi + \sum_{v \in \mathcal{X}} min(clause\_unitaire(v), clause\_unitaire(\neg v))$$

où $\phi$ est la formule associée à l'affectation courante. Pour intégrer $LB2$ à l'algorithme 1, la condition de la ligne 5 doit être remplacée par $LB2(\phi) \geq ub$.

Les mécanismes tels que $UP$ et $LB2$ accélère la recherche mais l'un des composants les plus importants d'un algorithme de type B&B (comme pour les algorithmes de type DPLL) est son heuristique de branchement.

## 3 Les Heuristiques de Branchement

Une heuristique de branchement sélectionne la prochaine variable allant être valuée. Une bonne heuristique doit permettre à l'algorithme de rapidement faire diminuer sa valeur $ub$ et ainsi couper de nombreuses branches de l'arbre de recherche. Plusieurs heuristiques ont été proposées pour SAT [8, 7, 4, 10, 21] fournissant toutes de bons résultats. Par contre, concernant le problème MAX-SAT, très peu d'heuristiques spécifiques ont réellement été proposées. Ce sont donc deux heuristiques définies pour le problème SAT (MOMS et JW) qui sont le plus souvent utilisées dans les algorithmes pour MAX-SAT. Une troisième heuristique, appelée Dynamic-weighted, peut aussi être utilisée mais elle ne fait que choisir quelle heuristique, MOMS ou JW, il est le plus intéressant d'appliquer à chaque nœud de l'arbre de recherche.

### 3.1 Heuristique MOMS

L'heuristique MOMS (Maximum Occurrences in clauses of Minimum Size)[4] est une des heuristiques les plus simples : elle sélectionne la variable ayant le plus d'occurrences dans les clauses de plus petite taille.

> **Exemple 1 (Heuristique MOMS) :**
> Soit une formule $\phi$: $(\neg a \vee b) \wedge (\neg b \vee d \vee \neg c) \wedge (a \vee \neg e) \wedge (b \vee d \vee f) \wedge (\neg b \vee e \vee f) \wedge (b \vee \neg e \vee c)$
> Les clauses les plus petites sont $(\neg a \vee b)$ et $(a \vee \neg e)$ donc l'heuristique MOMS sélectionnera la variable $a$.

### 3.2 Heuristique Jeroslow-Wang

L'heuristique Jeroslow-Wang (JW) [8] fonctionne d'une manière similaire à l'heuristique MOMS excepté que la variable n'est pas sélectionnée parmi les clauses de plus petite taille mais parmi toutes les clauses. Un poids est donné pour chaque littéral en fonction de la taille des clauses où il apparaît. JW utilise une fonction $J$ qui prend en entrée un littéral $l$ et retourne un poids pour ce littéral.

$$J(l) = \sum_{l \in c \in \mathcal{C}} 2^{-|c|}$$

où $|c|$ est la taille de la clause $c$. La variable $v$ sélectionnée par l'heuristique JW est celle qui maximise $J(v) + J(\neg v)$.

> **Exemple 2 (Heuristique JW) :**
> Soit la formule $\phi$: $(\neg a \vee b) \wedge (\neg b \vee d \vee \neg c) \wedge (a \vee \neg e) \wedge (b \vee d \vee f) \wedge (\neg b \vee e \vee f) \wedge (b \vee \neg e \vee c)$
> Avec la fonction $J$ : $J(a) = 0.25$, $J(\neg a) = 0.25$, $J(b) = 0.5$, $J(\neg b) = 0.25$, $J(c) = 0.125$, $J(\neg c) = 0.125$, $J(d) = 0.25$, $J(\neg d) = 0$, $J(e) = 0.125$, $J(\neg e) = 0.375$, $J(f) = 0.25$, $J(\neg f) = 0$.
> La variable $b$ avec $J(b) + J(\neg b) = 0.75$ est sélectionnée par JW.

### 3.3 Heuristique Dynamic-weighting

Les résultats expérimentaux présentés dans [20] montrent que l'utilisation de l'heuristique JW (resp. MOMS) diminue, pour certaines valeurs du rapport clauses/variables, le nombre de backtracks par rapport à l'heuristique MOMS (resp. JW). Il a donc été proposé une heuristique qui sélectionne soit l'heuristique MOMS soit l'heuristique JW à chaque nœud en fonction du rapport clauses/variables :

- si $clauses/variables < 6.3$ alors le branchement est effectué par MOMS;
- si $6.3 \leq clauses/variables \leq 7.2$ alors le branchement est effectué par JW mais avec $J(l) = \sum_{\{c \in \mathcal{C} | l \in c\}} (26 - 3.33(clauses/variables))^{-|c|}$;
- si $clauses/variables > 7.2$ alors le branchement est effectué par JW;

## 4 Comportements des Heuristiques de Branchement

La plupart des algorithmes de type B&B pour MAX-SAT [2, 1, 22, 20] démarrent leur recherche avec le nombre optimal de clauses vides. Ceci est principalement dû au fait qu'ils utilisent un algorithme de recherche locale pour fournir la borne supérieure initiale et que, pour de petites instances, il est facile de trouver la borne supérieure optimale avec une recherche locale. Bien sur, pour de grandes instances (supérieures à 1000 variables) cela devient beaucoup plus difficile.

Notre étude du comportement des heuristiques consiste à observer ce qui se passe si la borne supérieure initiale ($iub$) n'est pas optimale. Les tests ont été réalisés sur les instances proposé par Borchers et Furman [2] qui sont utilisées dans [2, 1, 22, 20]. Nos résultats sont présentés pour des instances 3SAT aléatoires ayant 50 variables et dont le nombre de clauses varie entre 250 et 500. Il est bien connu que les instances 3SAT aléatoires provoquent un phénomène de transition de phase quand le rapport clauses/variables est égal à 4.26 [13] : pour des rapports plus élevés, les instances 3SAT aléatoires ont une forte probabilité d'être insatisfiable (et donc d'être intéressante pour le problème MAX-SAT), alors que pour des rapports plus petits, il est très facile de trouver des solutions qui satisfassent toutes les clauses. Pour cette raison, toutes les instances utilisées pour les tests possèdent un rapport supérieur ou égal à 5. Les comparaisons entre MOMS, JW et Dynamic-weighting sont effectuées avec et sans la propagation unitaire et l'estimation de la borne inférieure. Par conséquent, ceci nous amène à considérer quatre algorithmes.

### 4.1 Résultats Expérimentaux

Les figures 1, 2, 3 et 4 représentent l'évolution de l'effort de recherche du Branch and Bound (nombre de backtracks) en fonction de la valeur de la borne supérieure initiale ($iub$). Pour chaque instance, la $iub$ varie de la borne supérieure optimale ($bub$) à $bub + 9$ par pas de 1 (i.e. 10 valeurs de borne initiale différentes).

### 4.2 Analyses

Les figures 1, 2, 3 et 4 montrent que les heuristiques MOMS, JW et Dynamic-weighting ont des comportements différents en fonction de la valeur de la borne supérieure initiale $iub$. Elles montrent aussi que les mécanismes de propagation unitaire et d'estimation de borne inférieure n'ont pas les mêmes effets en fonction de l'heuristique choisie.

Les figures 1 et 2 mettent en avant que l'utilisation de l'heuristique MOMS avec ou sans propagation unitaire et sans estimation de la borne inférieure fournit toujours de meilleurs résultats quand elle démarre avec la borne optimale comme borne initiale. Pour l'instance avec un fort rapport clauses/variables (500 clauses), l'heuristique JW devient plus efficace quand la $iub$ est élevée. Au contraire, pour les autres instances (250 et 400 clauses), plus la $iub$ est élevée, meilleurs sont les résultats de l'heuristique MOMS par rapport à ceux de l'heuristique JW. Pour ces deux figures, l'heuristique Dynamic-weighting apparaît toujours meilleure que les autres. Pour les trois heuristiques, l'utilisation de la propagation unitaire améliore tous les résultats obtenus figure 1.

La figure 3 montre que JW est généralement meilleure que MOMS et Dynamic-weighting. Il est intéressant de remarquer que pour la plus petite instance, les résultats de

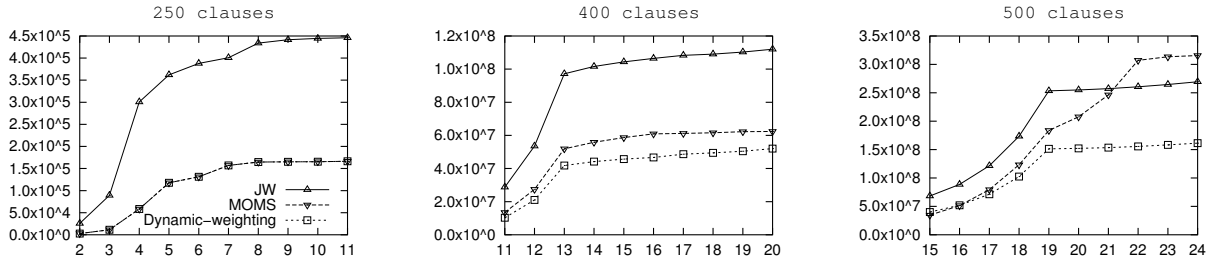


FIG. 1 – *Tests réalisés sans les mécanismes UP et LB2. Les abscisses représentent la borne supérieure initiale et les ordonnées le nombre de backtracks.*

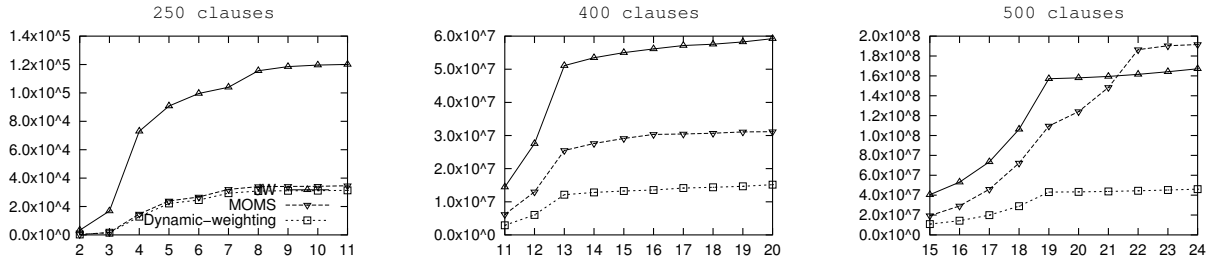


FIG. 2 – *Tests réalisés avec le mécanisme UP mais sans le mécanisme LB2. Les abscisses représentent la borne supérieure initiale et les ordonnées le nombre de backtracks.*

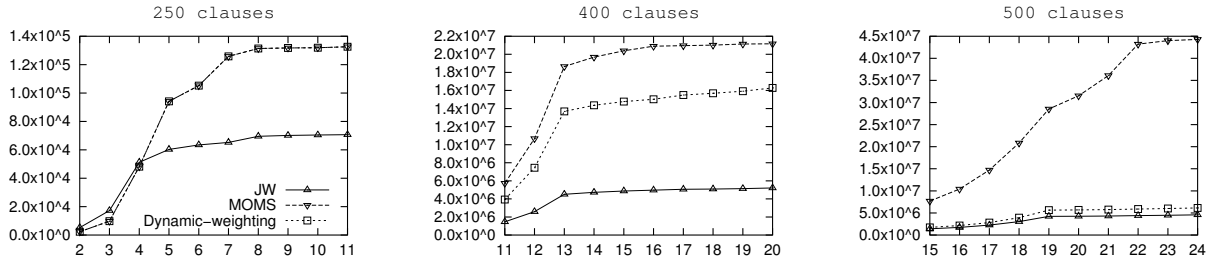


FIG. 3 – *Tests réalisés sans le mécanisme UP mais avec le mécanisme LB2. Les abscisses représentent la borne supérieure initiale et les ordonnées le nombre de backtracks.*

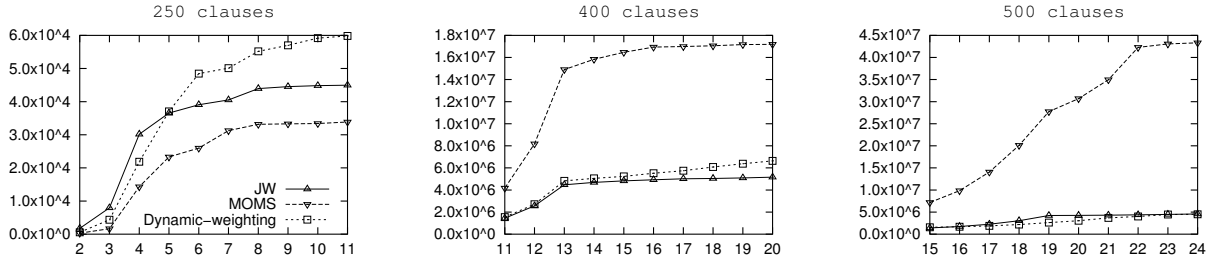


FIG. 4 – *Tests réalisés avec les mécanismes UP et LB2. Les abscisses représentent la borne supérieure initiale et les ordonnées le nombre de backtracks.*

JW sont plus mauvais pour une $iub$ proche de $bub$ que pour une $iub$ élevée. Ces résultats montrent que la valeur de $iub$ joue un rôle important dans le comportement des heuristiques. Contrairement à la propagation unitaire, l'utilisation de l'estimation de la borne inférieure améliore énormément les résultats de JW alors que ceux de MOMS ne le sont que faiblement. Le comportement de l'heuristique Dynamic-weighting est similaire à celui de MOMS pour les instances avec un faible rapport clauses/variables alors que pour un rapport élevé, il est similaire à JW.

La figure 4 présente les résultats des heuristiques avec les deux mécanismes. Pour la plus petite instance, MOMS obtient les meilleurs résultats mais pour les autres, JW fournit d'excellents résultats par rapport à MOMS. Il est à noter que, pour les deux grandes instances, le nombre de backtracks semble être constant pour JW alors qu'il croît en fonction de $iub$ pour MOMS. Tout comme dans la figure 3, le comportement de l'heuristique Dynamic-weighting est similaire à MOMS ou JW excepté pour la plus petite instance où le nombre de backtrack croît très rapidement.

**Synthèses:**

D'après les quatre figures, plusieurs points sont ressortis :

- UP améliore les performances de toutes les heuristiques de manière identique;
- LB2 améliore principalement les performances de JW;
- quand UP et LB2 sont combinés, leurs effets sont complémentaires.

D'autres expériences ont été menées avec d'autres instances 3SAT aléatoires. Des résultats similaires ont été obtenus pour des instances ayant des nombres de variables différents.

## 5 Une Autre Hybridation des Heuristiques MOMS et JW

Les heuristiques MOMS et JW fournissent pour chaque nœud de l'arbre de recherche un ordre sur les variables non instanciées. On peut alors se demander ce qui se produit quand l'information fournie par ces deux heuristiques est réellement combinée et non choisie individuellement comme dans l'heuristique Dynamic-weighting. Dans cette partie, nous étudions le comportement d'une telle combinaison d'heuristiques (MOMS-JW). En fait, cette combinaison définit un nouvel ordre en faisant une moyenne des positions des variables dans les ordres induits par MOMS et JW. La variable ayant la meilleure moyenne est sélectionnée.

$$MOMS - JW(\phi) = x \text{ tel que } \forall y \in \mathcal{X}$$
$$r_{MOMS}(x) + r_{JW}(x) \leq r_{MOMS}(y) + r_{JW}(y)$$

où $r_h(x)$ retourne le rang de la variable $x$ dans l'ordre fourni par l'heuristique $h$.

**Exemple 3 (Heuristique MOMS-JW) :**
Soit la formule $\phi$: $(\neg a \vee b) \wedge (\neg b \vee d \vee e) \wedge (a \vee \neg e) \wedge (b \vee d \vee f) \wedge (\neg b \vee e \vee f) \wedge (b \vee \neg e \vee c)$
Avec l'heuristique MOMS, l'ordre est : $a, b, e$.
Avec l'heuristique JW, l'ordre est : $b, e, a, d, c, f$.
La variable retournée par MOMS-JW est donc $b$.

Nous avons testé l'heuristique MOMS-JW avec le même processus expérimental et sur les mêmes instances que la section précédente. Ces résultats ont été comparés avec ceux obtenus par MOMS, JW et Dynamic-weighting. Nous avons remarqué que pour toutes les instances, le comportement de MOMS-JW est plus influencé par JW et plus spécialement quand la propagation unitaire n'est pas utilisée. Avec l'estimation de la borne inférieure, ces résultats sont très compétitifs et, quand la propagation unitaire n'est pas utilisée, ils améliorent ceux de l'heuristique Dynamic-weighting. Les meilleurs résultats sont obtenus pour la plus petite instance avec la propagation unitaire ainsi que l'estimation de la borne inférieure (Figure 5) pour laquelle MOMS-JW améliore toutes les autres heuristiques. Il est intéressant de remarquer que pour les grandes instances (400 et 500 clauses), l'utilisation de la propagation unitaire détériore les résultats de MOMS-JW alors que ceux de MOMS et Dynamic-weighting sont améliorés et ceux de JW ne changent pas.

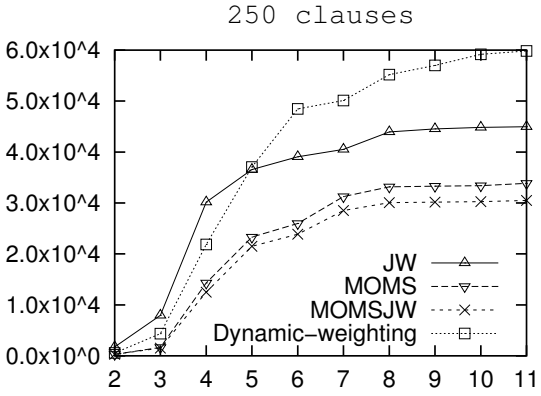


FIG. 5 – *Tests réalisés avec les mécanismes UP et LB2. Les abscisses représentent la borne supérieure initiale et les ordonnées le nombre de backtracks.*

Cette combinaison très simple de MOMS et JW peut fournir des résultats compétitifs. Il semble donc possible de proposer des heuristiques spécifiques au problème MAX-SAT améliorant les performances des heuristiques venant du problème SAT.

## 6 Conclusion

Dans ce papier nous avons effectué une étude de deux heuristiques de branchement bien connues appliquées au problème MAX-SAT. Nous avons montré que ces deux heuristiques fournissent de bons résultats mais que leur comportement est très dépendant de la valeur de la borne supérieure initiale et du choix des mécanismes. La combinaison de ces heuristiques proposée dans [20] est très compétitive mais elle reste dépendante de la borne supérieure initiale.

Tous les tests réalisés par les algorithmes complets pour le problème MAX-SAT sont faits sur de petites instances et la recherche démarre toujours avec la borne supérieure optimale. En effet, l'algorithme approché fournissant la borne supérieure initiale trouve très facilement la borne supérieure optimale. Nous avons effectué de nombreuses expérimentations avec différentes valeurs de borne initiale montrant ainsi que le comportement de ces heuristiques est très différent pour des valeurs de bornes initiales non optimales.

Pour compléter notre étude, nous avons testé une autre combinaison des heuristiques MOMS et JW. C'est une combinaison très simple mais permettant d'obtenir des résultats intéressants. Nos projets futurs sont de proposer de nouvelles heuristiques propres au problème MAX-SAT.

## Références


[1] Teresa Alsinet, Felip Manyà, and Jordi Planes. Improved branch and bound algorithms for MAX-SAT. In *Proc of the 6th International Conference on the Theory and Applications of Satisfiability Testing. SAT2003*, pages 408–415, 2003.

[2] Brian Borchers and Judith Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2:299–306, 1999.

[3] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[4] Olivier Dubois, Pascal André, Yacine Boufkhad, and Jacques Carlier. Can a very simple algorithm be efficient for solving the SAT problem? In *Proc. of the DIMACS Challenge II Workshop*, 1993.

[5] Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In Bernhard Nebel, editor, *Proc. of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 248–253, San Francisco, CA, 2001.

[6] Michael R. Garey and David S. Johnson. *Computers and Intractability , A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, San Francisco, 1979.

[7] John N. Hooker and V. Vinay. Branching rules for satisfiability. *Journal of Automated Reasoning*, 15(3):359–383, Dec 1995.

[8] Robert G. Jeroslow and Jinchang Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[9] Steve Joy, Brian Borchers, and John E. Mitchell. A branch-and-cut algorithm for MAX-SAT and weighted MAX-SAT. In D. Du, J. Gu, and P. M. Pardalos, editors, *Satisfiability Problem: Theory and Applications*, volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 519–536. AMS/DIMACS, 1997.

[10] Chu Min Li and Anbulagan Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proc. of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 366–371, 1997.

[11] Donald Loveland. *Automatic Theorem Proving*. Elsevier Science Publishers North-Holland, 1978.

[12] João Marques Silva and Karem. A. Sakallah. GRASP: A new search algorithm for satisfiability. In *Proc. of the IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, Washington, 1996.

[13] David G. Mitchell, Bart Selman, and Hector J. Levesque. Hard and easy distributions for SAT problems. In *Proc. of AAAI'92*, pages 459–465, 1992.

[14] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proc. of the 38th Design Automation Conference (DAC'01)*, Jun 2001.

[15] Richard Ostrowski, Eric Grégoire, Bertrand Mazure, and Lakhdar Sais. Recovering and exploiting structural knowledge from cnf formulas. In *Proc. of the 8th International Conference of Constraint Programming (CP'02)*, volume 2470 of *Lecture Notes in Computer Science*, pages 185–199. Springer, 2002.

[16] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proc. of the AAAI'94, Vol. 1*, pages 337–343, 1994.

[17] Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In *Proc. of the AAAI'92*, pages 440–446, San Jose, CA, 1992.

[18] Richard J. Wallace and Eugene C. Freuder. Comparative studies of constraint satisfaction and Davis-Putnam algorithms for maximum satisfiability problems. In *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 587–615, 1996.

[19] Zhao Xing and Weixiong Zhang. Efficient strategies for (weighted) maximum satisfiability. In *Proc of the 10th International Conference on Constraint Programming (CP'04)*, volume 3258 of *Lecture Notes in Computer Science*, pages 690–705. Springer, 2004.

[20] Zhao Xing and Weixiong Zhang. Maxsolver: An efficient exact algorithm for (weighted) maximum satisfiability. *Artificial intelligence*, 164(1–2):47–80, 2005.

[21] Hantao Zhang. SATO: An efficient propositional prover. In *Proc. of the 14th International Conference on Automated Deduction*, volume 1249 of *LNAI*, pages 272–275, Berlin, 1997.

[22] Hantao Zhang, Haiou Shen, and Felip Manyà. Exact algorithms for MAX-SAT. In Ingo Dahn and Laurent Vigneron, editors, *Electronic Notes in Theoretical Computer Science*, volume 86. Elsevier, 2003.